PRÁCTICA 1
CENTRO VETERINARIO: UNMASCOTA

ASIGNATURA
Programación orientada a objetos

GRUPO
Grupo 2

EQUIPO
Equipo 6

PROFESOR
Jaime Alberto Guzmán Luna

INTEGRANTES

López González Alejandro

Betancur Uribe Emmanuel

Martínez Ríos Santiago

Bula Fuente Melanie

Ospina Gaviria Tomas

Universidad Nacional de Colombia

Sede Medellín

2025

Índice

1. Portada

Nombre actividad: Práctica 1

- **Nombre proyecto:** Centro Veterinario: UNamascota
- **Materia:** Programación Orientada a Objetos
- **Grupo:** 02
- **Equipo:** 6
- **Integrantes:** López González Alejandro, Betancur Uribe Emmanuel, Martínez Ríos Santiago, Bula Fuente, Melanie, Ospina Gaviria Tomas
- **Profesor:** Jaime Alberto Guzmán Luna
- **Nombre de la universidad:** Universidad Nacional de Colombia - Sede Medellín
- **Fecha:** 2025

2. Descripción general de la solución.

El análisis del dominio del proyecto es el primer paso en el desarrollo de la aplicación. En esta fase, se identifican y definen los requisitos y necesidades del proyecto. Este análisis incluye:

Centro Veterinario: UNamascota

UNamascota es un proyecto innovador que busca promover el cuidado responsable de animales domésticos y mejorar el bienestar animal en Medellín y sus alrededores. Con sedes estratégicamente ubicadas, UNamascota ofrece una plataforma integral para facilitar el proceso de cuidado animal y brindar servicios complementarios para el cuidado de las mascotas.

Objetivos Clave

- Facilitar el proceso de cuidado veterinario, asegurando que los animales estén en óptimas condiciones de salud y brindando una correcta retroalimentación sobre el estado de las mascotas.
- Promover la tenencia responsable de mascotas a través de educación y acompañamiento.
- Ofrecer una variedad de servicios para el cuidado integral de las mascotas, desde atención veterinaria hasta guardería y peluquería.
- Proveer productos de calidad para el bienestar de los animales en la tienda del proyecto.
- Ofrecer servicios de cremación y entierro digno para mascotas fallecidas, brindando un espacio de recuerdo y despedida.

Servicios Ofrecidos

1. **Emergencia Veterinaria:** Los usuarios pueden visitar cualquiera de las tres sedes de UNamascota para conocer el estado de sus mascotas y determinar si requieren hospitalización.
2. **Agendamiento de Servicios:** Los usuarios pueden reservar citas con especialistas en diferentes áreas a través de la plataforma de UNamascota.
3. **Funeraria:** En caso de fallecimiento de una mascota, UNamascota ofrece servicios de cremación y entierro, así como un espacio para que los usuarios puedan visitar y dejar flores en honor a su compañero.
4. **Tienda:** UNamascota cuenta con una tienda donde los usuarios pueden adquirir productos de calidad para el cuidado y bienestar de sus mascotas.
5. **Dieta Personalizada:** Muchos animales tienen necesidades específicas según sus características; por ello, planificar una dieta adecuada y conocer el estado nutricional de tu mascota es de suma importancia.

Objetivo del Centro Veterinario: UNamascota

UNamascota está diseñado para abordar y resolver diversas necesidades en el ámbito del cuidado responsable de mascotas y el bienestar animal. Su implementación tiene como finalidad facilitar la interacción entre los dueños y los animales, así como optimizar los servicios ofrecidos por el centro veterinario.

Necesidades a Solucionar

Optimización del Proceso de Emergencia Veterinaria:

- **Problema:** Los dueños suelen tener dificultades para identificar enfermedades en sus mascotas.
- **Solución:** El software proporcionará una interfaz amigable para que los usuarios puedan reportar los síntomas de sus mascotas. El sistema calculará la gravedad de la situación y determinará si es necesario hospitalizar al animal.

Centralización de Servicios para el Bienestar de las Mascotas:

- **Problema:** Los dueños necesitan acceder a múltiples servicios de manera eficiente para el cuidado de sus mascotas.
- **Solución:** El sistema permitirá agendar citas y acceder a servicios complementarios desde una única plataforma, mejorando la experiencia y comodidad del usuario.

Gestión de Servicios Funerarios:

- **Problema:** La pérdida de una mascota es un momento difícil, y los dueños requieren opciones seguras y de calidad para despedirse dignamente.
- **Solución:** El software ofrecerá información sobre servicios funerarios, incluyendo cremación y entierro, y permitirá a los usuarios gestionar estos servicios de manera sencilla.

Integración de Tienda en Línea:

- **Problema:** Los dueños suelen enfrentarse a catálogos extensos de productos, sin saber cuáles son los más adecuados para sus mascotas.
- **Solución:** Los usuarios podrán adquirir productos esenciales para sus mascotas utilizando filtros que faciliten encontrar alimentos, juguetes, accesorios e insumos para el hogar, todo centralizado en la misma plataforma que gestiona los servicios veterinarios y complementarios.

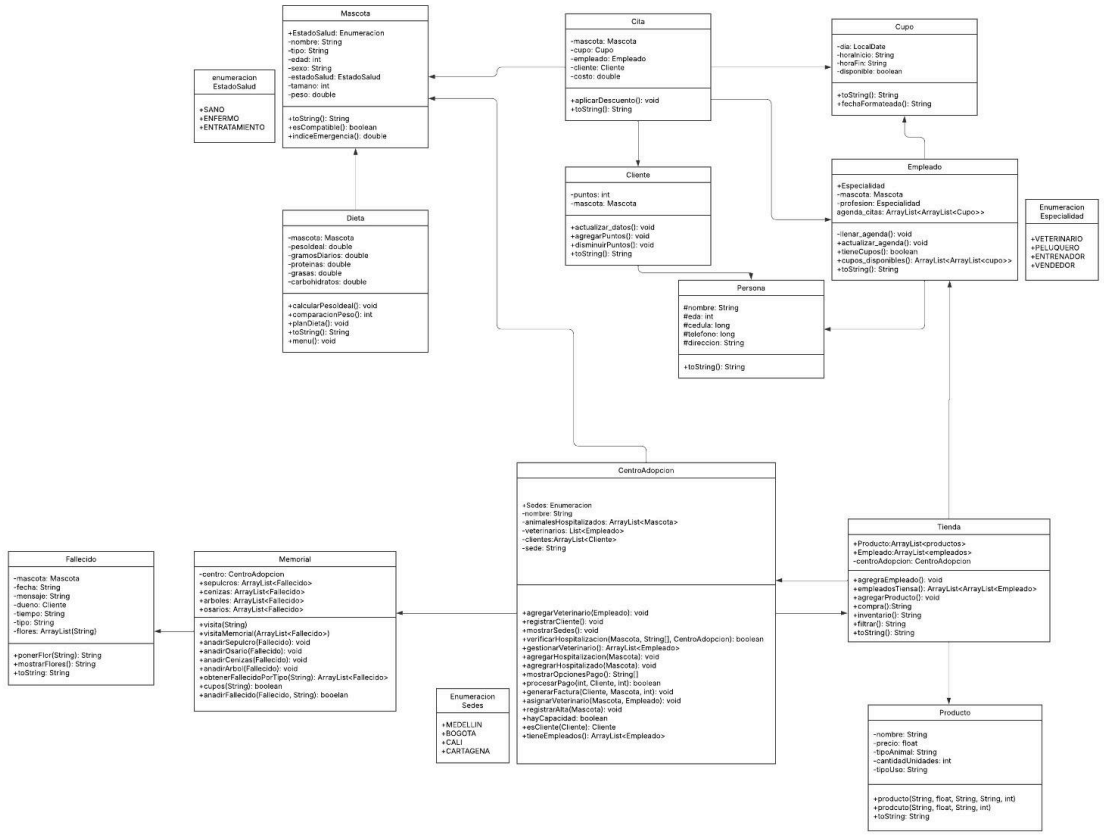
Sistema de Planificación de Dietas:

- **Problema:** Los dueños de mascotas suelen descuidar el aspecto de la salud nutricional de sus mascotas.
- **Solución:** Desde UNamascota se puede hacer un análisis de las condiciones nutricionales de su mascota para comprender la solución pertinente que se debe implementar respecto a su alimentación.

3. Descripción del diseño estático del sistema en la especificación UML.

En esta sección, se debe incluir el diagrama de clases que representa el dominio de la aplicación. Este diagrama debe mostrar todas las clases principales del sistema junto con las relaciones de cardinalidad entre ellas. La cardinalidad indica cuántas instancias de una clase pueden estar asociadas con una instancia de otra clase, proporcionando una visión clara de las interacciones y dependencias dentro del sistema.

Además del diagrama de clases, es importante proporcionar una breve descripción de la función de cada una de las clases en el proyecto. Esta descripción debe incluir:



1. CentroAdopcion

Propósito:
Hace referencia a las sedes, gestiona la operación del centro de adopción, incluyendo clientes, empleados, animales, citas y adopciones. Facilita el proceso de adopción y la prestación de servicios relacionados.

Relaciones:

- Cero a muchos con **Empleado**.
- Cero a muchos con **Mascota**.
- Cero a muchos con **Cliente**.
- Cero a muchos con **Cita**.
- Cero a uno con **Tienda**.
- Cero a uno con **Memorial**.

2. Mascota

Propósito:
Representa a los animales disponibles para adopción. Incluye atributos como nombre, tipo, edad, sexo y estado de salud. También representa a las mascotas que un usuario registre para recibir un servicio o agendar una cita.

3. Persona

Propósito:
Representa a las personas relacionadas con el centro de adopción, ya sean clientes o empleados. Almacena información personal básica.

Relaciones:

- Es una superclase abstracta de la que heredan **Cliente** y **Empleado**.

4. Empleado

Propósito:
Representa al personal del centro, que puede ser veterinario, peluquero, cuidador o tendero. Gestiona la atención a los clientes y animales.

Relaciones:

- Uno a muchos con **Cupo**.

5. Cupo

Propósito:
Representa un bloque de tiempo disponible para citas asignado a cada empleado. Incluye información sobre el día, la hora de inicio y fin, y si está disponible.

Relaciones:

- Uno a uno con **Cita** (cada cita utiliza un único cupo).

6. Cliente

Propósito:
Representa a los usuarios que buscan adoptar animales y utilizar otros servicios del centro, como la tienda y citas.

Relaciones:

- Uno a muchos con Mascota (Un Cliente puede tener múltiples mascotas).
- Uno a muchos con **Cita** (un cliente puede agendar múltiples citas).

7. Cita

Propósito:
Almacena información sobre las citas agendadas para servicios como veterinaria, guardería o peluquería. Incluye detalles del animal, cliente, empleado y costo.

Relaciones:

- Uno a uno con Mascota, **Cliente** y **Empleado** (cada cita está asociada a un único animal, cliente y

empleado).
- Uno a uno con **Cupo** (cada cita ocupa un cupo específico).

8. Producto

Propósito:
Representa los artículos en venta en la tienda, incluyendo nombre, precio, tipo de animal y cantidad disponible.

Relaciones:

- No tiene relaciones directas con otras clases, pero es referenciado por la clase **Tienda**.

9. Tienda

Propósito:
Gestiona el inventario de productos disponibles para la venta, permitiendo a los clientes realizar compras.

Relaciones:

- Uno a muchos con **Producto** (una tienda puede tener múltiples productos).
- Uno a muchos con **Empleado** (los empleados pueden gestionar la tienda).

10. Memorial

Propósito:
Proporciona servicios de cremación y entierro para mascotas fallecidas, gestionando la memoria de los animales.

Relaciones:

- Uno a muchos con **Fallecido** (puede haber múltiples registros de mascotas fallecidas).

11. Fallecido

Propósito:
Representa a los animales que han fallecido, almacenando información sobre el dueño, fecha de fallecimiento y mensajes de recuerdo.

Relaciones:

- Uno a uno con **Mascota**(cada registro de "Fallecido" se refiere a un único animal fallecido).
- Uno a uno con **Cliente** (cada registro está asociado a un único dueño).

12. Dieta

Propósito:
Representa los requerimientos e información nutricional de cada mascota usado para recetar diferentes alimentaciones especiales.

Relaciones:

- Uno a uno con **Mascota** (Una mascota tiene una única dieta)

4. Descripción de la implementación de características de programación orientada a objetos en el proyecto.

En el enunciado de la práctica se solicitó la implementación de varias características de programación orientada a objetos.

- Clase Abstracta y Métodos Abstractos:

gestorAplicacion.elementos.Persona (Línea 11)

```
11  public abstract class Persona implements Serializable {
12      private static final long serialVersionUID = 1L;
13      //---> Atributos <---
14      protected String nombre;
15      protected int edad;
16      protected long cedula;
17      protected long telefono;
18      protected String direccion;
19
```

Se decidió utilizar la clase Persona como clase abstracta, debido a que como persona no interviene directamente con la creación de objetos y cumple más un papel de guía para las subclases que heredan de ella y asÍ hay una mejor optimización de recursos.

- Herencia:

gestorAplicacion.elementos.Cliente (Línea 17)

```
17  public class Cliente extends Persona implements Serializable {
18
19      private static final long serialVersionUID = 1L;
20
21      //ATRIBUTOS
22
23      private int puntos;
24      private Mascota mascota;
```

```
13  public class Empleado extends Persona implements Serializable{
14      private static final long serialVersionUID = 1L;
15      public static enum Especialidad {
16          VETERINARIO,PELUQUERO,ENTRENADOR,VENDEDOR;
17      }
18
```

Se decidió hacer uso de la herencia en las clases Cliente, Empleado y Persona, de manera que Persona herede a Cliente y Empleado, para reutilizar atributos que tienen tanto la clase Empleado como la Cliente, y así optimizar mejor nuestras líneas de código.

- Ligadura Dinámica:

  gestionAplicacion.elementos.Persona (Linea 72)

```
71      //---> Metodo ToString <---
72      public String toString() {
73          return "Nombre: " + getNombre() + "\nEdad: "+ getEdad()+ "\nCedula: " + getCedula() +"\nTelefono: " + getTelefono() + "\nDirección "+ getDireccion() + "\n";
74      }
75  }
```

Si hay clases que heredan de Persona y sobrescriben métodos como toString(), entonces se aprovechará la ligadura dinámica para resolver esos métodos en tiempo de ejecución.

- Atributos de Clase Y Métodos de Clase:

  gestionAplicacion.gestion.Tienda (Linea 25)

  gestionAplicacion.gestion.Tienda (Linea 26)

  gestionAplicacion.gestion.Tienda (Linea 51)

  gestionAplicacion.gestion.Tienda (Linea 55)

```
51          public static ArrayList<Producto> getProductos(){
52              return productos;
53          }
54
55          public static ArrayList<Empleado> getEmpleados(){
56              return empleados;
57          }
```

```
22          private static final long serialVersionUID = 1L;
23          // ATRIBUTOS
24          //LISTAS STATIC PARA LA SERIALIZACION
25          public static ArrayList<Producto> productos = new ArrayList<> ();
26          public static ArrayList<Empleado> empleados = new ArrayList<> ();
27          private CentroAdopcion centroAdopcion;
```

Se definen las listas como estáticas porque es necesario que estén para todos los objetos creados, y poder hacer llamadas desde la clase como el que se aplican en el deserealizador.

- Constante:

  gestorAplicacion.elementos.cliente (Linea 21)

```
public class Cliente extends Persona implements Serializable {

    private static final long serialVersionUID = 1L;

    public static final int EDAD_MINIMA = 18;
```

Se aplica la EDAD_MINIMA como

- Encapsulamiento:

  gestionAplicacion.elementos.Persona (Línea 14 - 18)

```
11  public abstract class Persona implements Serializable {
12      private static final long serialVersionUID = 1L;
13      //---> Atributos <---
14      protected String nombre;
15      protected int edad;
16      protected long cedula;
17      protected long telefono;
18      protected String direccion;
```

- Sobrecarga de Métodos:

  gestorAplicacion.gestion.Tienda (Línea 116 y 160)

```java
public String compra(int indice, Cliente cliente){

    if (empleados!=null){ // SE COMPRUEBA SI HAY UN EMPLEADO PARA LA OPERACION DE LA TIENDA

        indice -= 1; //CONVERTIMOS EL NÚMERO A INDICE DE LISTA

        if (indice>=productos.size() || indice<0) { //COMPROBAMOS SI EL INDICE ES CORRECTO
            return "🚫¡Oh no!, verifica que el índice del producto sea el indicado, el centro UNamascota desconoce ese índice.🚫";
        }

        int cantidad = productos.get(indice).getCantidadUnidades(); // SE TOMA LA CANTIDAD DEL PRODUCTO

        if (cantidad!=0){// SI LA CANTIDAD ES DIFERENTE DE 0 SE HACE LA COMPRA
            cantidad--1;
            productos.get(indice).setCantidadUnidades(cantidad); // SE TRAE EL PRODUCTO DE LA LISTA DE PRODUCTOS
            String nombre = productos.get(indice).getNombre();
            String tipo =  productos.get(indice).getTipoAnimal();

            if (productos.get(indice).getCantidadUnidades()==0) {
                productos.remove(indice); // SI LAS UNIDADES QUEDAN EN CERO, SE ELIMINA EL PRODUCTO DE LA LISTA
            }

            cliente = CentroAdopcion.esCliente(cliente); // SE COMPRUEBA SI EL CLIENTE EXISTE
            float precio = productos.get(indice).getPrecio(); // SE GUARDA EL PRECIO PARA LA FUTURA FACTURA
            int puntos = cliente.getPuntos(); //SE TOMAN LOS PUNTOS DEL CLIENTE

            if (puntos>=15) {// SI TIENE MÁS DE 15 PUNTOS ACOMULADOS, SE LE HACE UN DESCUENTO AUTOMÁTICAMENTE
                cliente.disminuir_Puntos(puntos); // SE LE QUITAN LOS 15 PUNTOS
                precio -= precio*0.1; // SE MODIFICA EL PRECIO ORIGINAL
                // EN EL RETURN DEVOLVEMOS LO IMPORTADO
                return "----------------------------------------------\n"+"Muchas gracias por tu compra, adquiriste: "+nombre+" - Dirigido a: "+tipo+"\n"+"Total a pagar: "+precio
            }else {
                return "----------------------------------------------\n"+"Muchas gracias por tu compra, adquiriste: "+nombre+" - Dirigido a: "+tipo+"\n"+"Total a pagar: "+precio+
            }
        }
        else{
            return "🚫¡Oh no!, verifica la cantidad de unidades disponibles en la tienda y vuelve a intentarlo.🚫";
        }
    }else{
        return "🚫Lo lamentamos pero el centro UNamascota no tiene empleados disponibles para atender su solicitud.🚫";
    }
}
```

```java
public String compra(int indice, int unidades, Cliente cliente){

    if (empleados!=null){ // SE COMPRUEBA SI HAY UN EMPLEADO PARA LA OPERACION DE LA TIENDA

        indice -= 1; //CONVERTIMOS EL NÚMERO A INDICE

        if (indice>=productos.size() || indice<0) { //COMPROBAMOS SI EL INDICE ES CORRECTO
            return "🚫¡Oh no!, verifica que el índice del producto sea el indicado, el centro UNamascota desconoce ese índice.🚫";
        }
        int cantidad = productos.get(indice).getCantidadUnidades(); //GUARDAMOS LA CANTIDAD DE UNIDADES DEL PRODUCTO A COMPRAR

        if (cantidad!=0 && cantidad>=unidades){ //SI LA CANTIDAD INGRESADA ESTÁ DENTRO DEL RANGO INICIA LA OPERACION DE LA TIENDA

            cantidad-=unidades; // SE RESTA LAS UNIDADES QUE SE COMPRARON
            productos.get(indice).setCantidadUnidades(cantidad); // SE CAMBIA LAS UNIDADES RESTANTES EN EL PRODUCTO
            String nombre = productos.get(indice).getNombre();
            String tipo = productos.get(indice).getTipoAnimal();
            if (productos.get(indice).getCantidadUnidades()==0) {
                productos.remove(indice); // SI LA CANTIDAD LLEGA A 0, SE ELIMINA EL PRODUCTO

            }

            cliente = CentroAdopcion.esCliente(cliente); // SE COMPRUEBA SI EL CLIENTE EXISTE
            float precio = productos.get(indice).getPrecio(); // SE GUARDA EL PRECIO PARA GENERAR DESPUES FACTURA
            int puntos = cliente.getPuntos(); // SE TOMAN LOS PUNTOS

            if (puntos>=15) { // SI TIENE MÁS DE 15 PUNTOS ACOMULADOS, SE LE HACE UN DESCUENTO AUTOMÁTICAMENTE
                cliente.disminuir_Puntos(puntos); // SE RESTAN LOS 15 PUNTOS QUE SE USARON COMO DESCUENTO
                precio -= precio*unidades*0.1; // SE CALCULA EL PRECIO CON EL 10% DE DESCUENTO
                return "----------------------------------------------\n"+"Has comprado "+unidades+" unidades de: "+nombre+" - Dirigido a: "+tipo+"\n"+"Total a pagar: "+precio+"
            }else { // RETORNA UNICAMENTE EL PRECIO PORQUE NO TIENE LOS PUNTOS SUFUCIENTES
                return "----------------------------------------------\n"+"Has comprado "+unidades+" unidades de: "+nombre+" - Dirigido a: "+tipo+"\n"+"Total a pagar: "+precio*unid
            }
        }
        else{
            return "🚫¡Oh no!, verifica la cantidad de unidades disponibles en la tienda y vuelve a intentarlo.🚫";
        }
    }else{
        return "🚫Lo lamentamos pero el centro UNamascota no tiene empleados disponibles para atender su solicitud.🚫";
    }
}
```

gestorAplicacion.elementos.mascota.java (linea 113)

```java
public int getTamano(){
    return tamano;
}

public String getTamano(String string){ //Si al llamar al metodo se acompaña de una string, devolvera el atributo en formato string
    return switch (tamano) {
        case 1 -> "Miniatura";
        case 2 -> "Pequeño";
        case 3 -> "Mediano";
        case 4 -> "Grande";
        default -> "Mediano";
    };
}
```

- Manejo This:

  gestorAplicacion.elementos.dieta.java (linea 18)

```java
//Constructor
public Dieta(Mascota mascota) {
    this.mascota = mascota;
}
```

  gestorAplicacion.elementos.mascota.java (linea 145)

```java
public boolean esCompatible(Mascota mascota) {
    if (this.tipo != mascota.tipo && this.estadoSalud != mascota.estadoSalud) {
        return false;
    }
    return true;
}
```

- Enumeración:

  gestionAplicacion.elementos.CentroAdopcion (Línea 11)

```java
9   public class CentroAdopcion implements Serializable {
10      private static final long serialVersionUID = 1L;
11      public static enum Sedes {
12          MEDELLIN, BOGOTA, CALI, CARTAGENA
13      }
```

  gestionAplicacion.elementos.mascota.java (linea 18)

```java
public static enum EstadoSalud {
    SANO, ENFERMO, ENTRATAMIENTO
}
```

5. Descripción de cada una de las 5 funcionalidades implementadas.

- **Descripción de la funcionalidad:** Proporcione una descripción detallada y completa del comportamiento de cada funcionalidad. Asegúrese de explicar claramente cada una de las interacciones que componen la funcionalidad, describiendo cómo los diferentes componentes del sistema colaboran para lograr el objetivo de la funcionalidad.
- **Diagrama de interacción:** Incluya una imagen del diagrama de interacción correspondiente a cada funcionalidad. Asegúrese de que el diagrama sea claro y permita observar el comportamiento general de la funcionalidad, destacando las interacciones clave y los flujos de datos entre los componentes del sistema.

❖ FUNCIONALIDAD 1: Emergencia Veterinaria

La funcionalidad Emergencia Veterinaria comienza llamando el método registro() de la clase Cliente, el cual le pide al usuario un conjunto de datos como nombre, edad y cédula, para así crear un objeto de tipo Cliente que servirá como un identificador para el usuario. De la misma forma, el usuario ingresará las propiedades de su mascota como nombre, especie, edad, sexo, tamaño y peso, además de los síntomas que presenta la misma, así, se instanciará un objeto de tipo Mascota con dichos parámetros, el cual tendrá su atributo "estadoSalud" asignado como "enfermo" por defecto, reflejando la urgencia médica.

```
Cliente cliente = Cliente.registro();
```

```
public static Cliente registro(){
    ArrayList<Object> datos = Main.capturarDatosCliente();
    Cliente cliente = new Cliente((String) datos.get(0), (int) datos.get(1), (long) datos.get(2));
    cliente.agregarPuntos(50000);

    return cliente;

}
```

```
public static ArrayList<Object> capturarDatosCliente() {
    try {Scanner sc = new Scanner(System.in);
    System.out.println("---------------------");
    System.out.println("\n- Ingrese sus datos");
    System.out.print("\n- Nombre Completo: ");
    String nombreC = sc.nextLine();
    System.out.print("- Edad: ");
    int edadC = sc.nextInt();
    sc.nextLine();
    System.out.print("- Cédula: ");
    long cedula = sc.nextLong();
    sc.nextLine();
    System.out.println("\n---------------------");

    ArrayList<Object> datos = new ArrayList<>();
    datos.add(nombreC);
    datos.add(edadC);
    datos.add(cedula);

    return datos;

    } finally {}

}
```

```
Mascota mascota1 = new Mascota(nombreMascota, tipo, edadMascota, sexo, EstadoSalud.ENFERMO, tamano, peso);
```

A continuación, el sistema muestra las sedes disponibles mediante el método mostrarSedes() de la clase CentroAdopcion, el cual accede a su lista de sedes y las muestra al usuario. Una vez seleccionada la sede, el sistema valida su disponibilidad para atender la emergencia, esto se realiza con el método verificarHospitalizacion() de CentroAdopcion que:

- Comprueba si hay capacidad disponible en la sede, verificando que la cantidad de mascotas hospitalizadas sea menor a 10.

```
public boolean hayCapacidad() {
    if (animalesHospitalizados.size() >= 10) {
        return false;
    }
    return true;
}
```

- Utiliza el método gestionarVeterinario() de la clase CentroAdopcion para verificar que la sede tenga veterinarios disponibles, haciendo uso de tieneCupos de la clase Empleado.

```
public List<Empleado> gestionarVeterinario() {
    List<Empleado> disponibles = new ArrayList<>();
    for (Empleado veterinario : veterinarios) {
        if (veterinario.tieneCupos()) {
            disponibles.add(veterinario);
        }
    }
    return disponibles.isEmpty() ? null : disponibles;
}
```

- Utiliza el método esCompatible() de la clase Mascota para verificar que el nuevo animal no represente un riesgo para los animales ya hospitalizados, basándose en parámetros como tipo y estado de salud.

```
public boolean esCompatible(Mascota mascota) {
    if (this.tipo != mascota.tipo && this.estadoSalud != mascota.estadoSalud) {
        return false;
    }
    return true;
}
```

- Evalúa tres factores del animal para determinar si requiere hospitalización inmediata:
  - Gravedad de los síntomas (de 1 a 10).
  - Edad del animal.
  - Indicador de compatibilidad con otros animales (de 0 a 10).

  El sistema calcula un Índice de Emergencia (IE) con la siguiente fórmula:

  $$IE = (Gravedad \times 0.7) + (Vulnerabilidad \times 0.3) - (Compatibilidad \times 0.1)$$

```
public double indiceEmergencia(int gravedad, int compatibilidad) {
    double vulnerabilidad = 10/(1+Math.abs(edad-4));
    double ie = (gravedad*0.7) + (vulnerabilidad*0.3) + (compatibilidad*0.1);

    return ie;
}
```

  Donde:

  - Gravedad es un valor calculado a partir de los síntomas.
  - Vulnerabilidad es una función de la edad del animal:
    - Animales jóvenes (<1 año) y ancianos (>8 años) tienen valores altos de vulnerabilidad.
    - Vulnerabilidad = 10/(1+|Edad-4|) (edad más cercana a 4 es menos

vulnerable).

- Compatibilidad es un valor calculado a partir de los síntomas, mide el riesgo de interacción con otros animales (valor bajo indica menor riesgo).

Si el índice de emergencia (IE) supera el umbral con valor predeterminado de 7.0, el animal se hospitaliza.

```java
public boolean verificarHospitalizacion(Mascota mascota, String[] listaSintomas, CentroAdopcion centro) {

    int gravedad = 0;
    int compatibilidad = 0;

    //Asignar valores de gravedad y compatibilidad a los síntomas
    for (String sintoma : listaSintomas) {
        switch (sintoma.toLowerCase()) {
            case "fiebre":
                gravedad += 2;
                compatibilidad += 3;
                break;
            case "vomito":
            case "vómito":
                gravedad += 3;
                compatibilidad += 2;
                break;
            case "picazon":
            case "picazón":
                gravedad += 2;
                compatibilidad += 1;
                break;
            case "enrojecimiento":
                gravedad += 1;
                compatibilidad += 2;
                break;
            case "inflamacion":
            case "inflamación":
                gravedad += 2;
                compatibilidad += 2;
                break;
            case "y":
                break;
            default:
                //System.out.println("\nSintoma desconocido: " + sintoma);
                break;
        }
    }

    if (mascota.indiceEmergencia(gravedad, compatibilidad) < 7.0) {
        return false;
    }
```

```java
    if (centro.gestionarVeterinario().isEmpty()) {
        return false;
    }

    if (!hayCapacidad()) {
        return false;
    }

    for (Mascota hospitalizado : animalesHospitalizados) {
        if (!mascota.esCompatible(hospitalizado)) {
            return false;
        }
    }
    return true;
}
```

Cuando se confirma si la mascota requiere hospitalización, se procede a listar los veterinarios disponibles. Esto se realiza mediante el método gestionarVeterinario() de la clase CentroAdopcion, que itera sobre la

lista de veterinarios de la sede y verifica su disponibilidad con el método tieneCupo() de la clase Empleado, y el método asignarVeterinario() de la clase CentroAdopcion, que asigna el veterinario que elige el usuario al atributo “veterinario” del objeto de tipo Mascota instanciado previamente, a su vez que éste objeto es asignado al atributo “mascota” de la instancia de Empleado.

```
public void asignarVeterinario(Mascota mascota, Empleado veterinario) {
    mascota.setVeterinario(veterinario);
    veterinario.setMascota(mascota);
}
```

Una vez asignado el veterinario que atenderá a la mascota, el sistema registra al animal en la lista de animales hospitalizados de la sede.

Cuando la atención médica o la hospitalización se han confirmado, el sistema gestiona el pago utilizando el método procesarPago() de CentroAdopcion. Las opciones disponibles incluyen: Tarjeta de crédito/débito, efectivo y/o puntos acumulados. Si el cliente elige pagar con puntos acumulados, el sistema valida el saldo de puntos disponibles mediante el método disminuirPuntos() de Cliente. Si los puntos no son suficientes, se solicita otro método de pago.

```
public boolean procesarPago(int metodo, Cliente cliente, int monto) {
    switch (metodo) {
        case 1:
            System.out.println("\nPago procesado con tarjeta por un monto de: $" + monto);
            return true;
        case 2:
            System.out.println("\nPago procesado en efectivo por un monto de: $" + monto);
            return true;
        case 3:
            if (cliente != null && monto <= cliente.getPuntos()) {
                cliente.disminuir_Puntos(monto);
                System.out.println("\nPago procesado con puntos acumulados.");
                return true;
            } else {
                System.out.println("\nNo tiene suficientes puntos.");
                return false;
            }
        default:
            System.out.println("\nMétodo de pago no válido.");
            return false;
    }
}
```

```
public void disminuir_Puntos(int puntos) {
    this.puntos-=puntos;
}
```

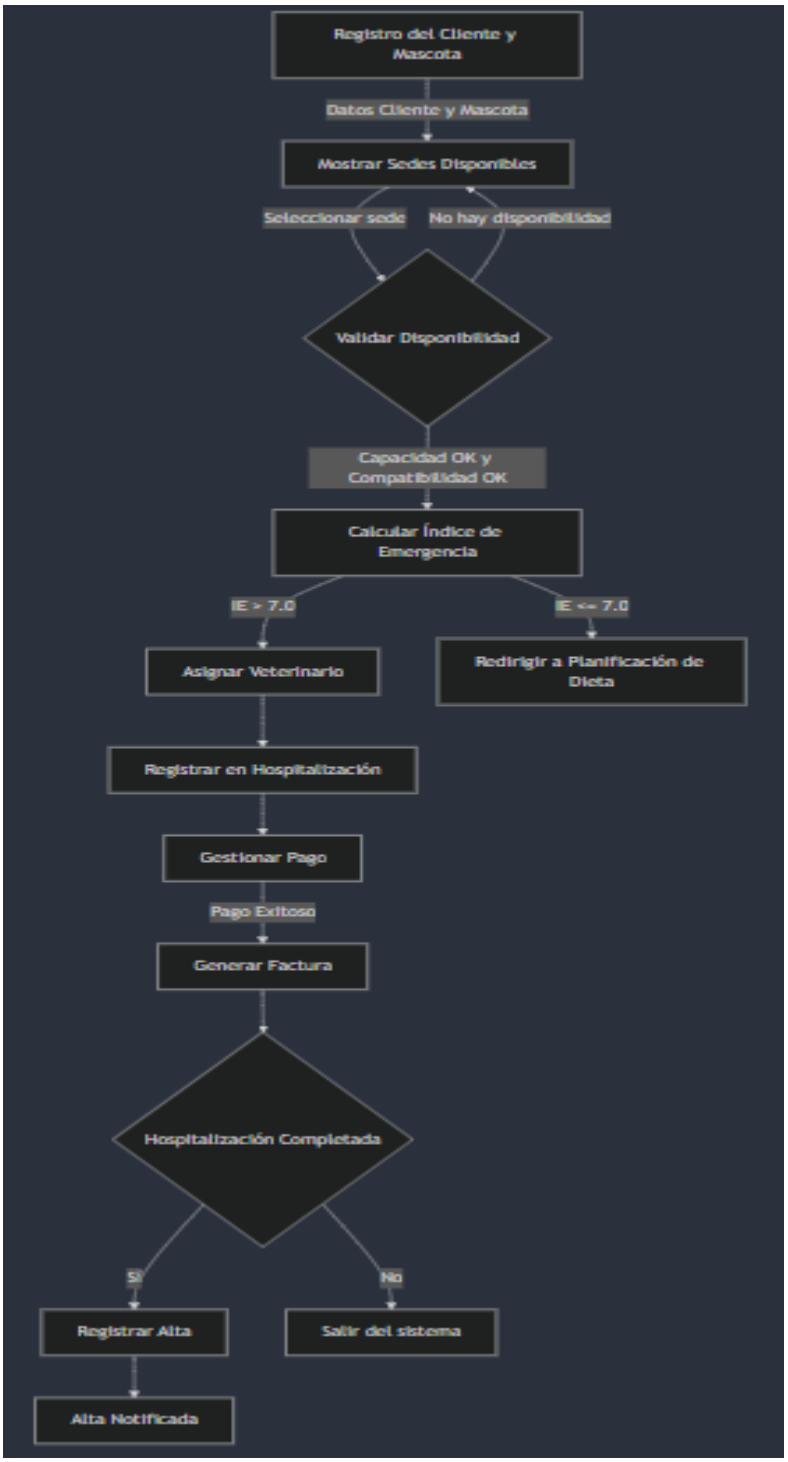
Al completar el pago, el sistema genera una factura detallada mediante el método generarFactura() de la clase CentroAdopcion. Esta factura incluye: Datos del cliente, información del animal y el monto total. La factura se presenta al usuario como comprobante del proceso.

```
public String generarFactura(Cliente cliente, Mascota mascota, int monto) {

    return "\n--------- Factura --------- " + "\n*|* Cliente     *|* " + (cliente != null ? cliente : "No registrado") +
           "\n*|* Animal      *|* " + mascota + "\n*|* Monto total *|* " + monto + "\n-----------------------------\n";

}
```

Finalmente, se le notificará al usuario que su mascota puede ser dada de alta, en caso de alta médica, el sistema permite registrar el alta a través del método registrarAlta(), el cual elimina la mascota de la lista de animales hospitalizados en el centro de adopción, cambia el atributo “estadoSalud” de la mascota a “sano”, y desvincula al veterinario de la mascota y viceversa. Por otro lado, si el animal no es hospitalizado, el usuario será redirigido a Planificación de Dieta.

```java
public void registrarAlta(Mascota mascota) {
    mascota.setEstadoSalud(EstadoSalud.SANO);
    animalesHospitalizados.remove(mascota);
    mascota.getVeterinario().setMascota(null);
    mascota.setVeterinario(null);
}
```

Diagrama de flujo



- **FUNCIONALIDAD 2: Agendar: UNServicio**

El Centro Veterinario Virtual UNamascota ofrece una funcionalidad para agendar UNServicio de forma semanal para diversos servicios, con el objetivo primordial de garantizar el bienestar y cuidado de las mascotas.

Esta herramienta permite a los usuarios solicitar y programar citas fácilmente para acceder a los servicios especializados que se ofrecen en cada una de las sedes.

Servicios ofrecidos

- En la sede Medellín, el servicio de Entrenamiento y Veterinaria.
- En la sede Bogotá, el servicio de Peluquería.
- En la sede de Cali, el servicio de Entrenamiento y Veterinaria.
- En la sede de Cartagena, el de Entrenamiento.

Los usuarios pueden agendar citas semanales con estos pasos:

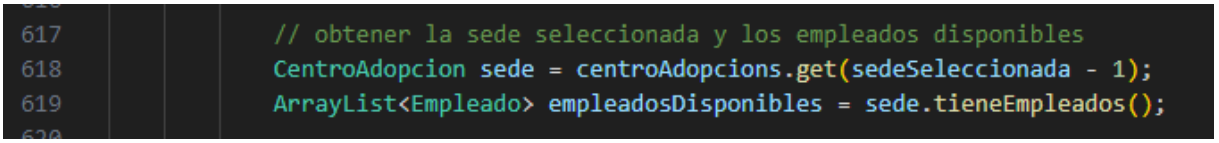
- Elegir la sede y el correspondiente servicio
- Seleccionar el día y horario disponible de la semana
- Proporcionar información básica del cliente y de la mascota.
- Confirmar la cita, que se agrega automáticamente al calendario

La agenda de citas se actualiza dinámicamente para evitar conflictos y permite un máximo de citas por día según la capacidad de cada sede.

La ejecución de la funcionalidad comienza con el cliente eligiendo la sede, dependiendo del servicio que requiera.

A continuación se le mostrará los tipos de mascotas para los cuales ese servicio está disponible; se le consultará si el tipo de su mascota, coincide con los ofertados; de no coincidir, se le informará que el proceso de agendar el servicio no podrá continuar.

De lo contrario; en primer lugar se verificará que la sede cuente con empleados con cupos disponibles en su agenda para atender a las mascotas, esto se ejecuta mediante el método tieneEmpleados() de la clase CentroAdopcion el cual retornará un ArrayList con los empleadoDisponibles; el método en su lógica recorrera el ArrayList de empleados de la sede selccionada y en cada Empleado ejecuta el método tieneCupos() de su misma clase.



```java
    //Se comprueba si tiene cupos dispponibles.
    public boolean tieneCupos() {

        Boolean booleano = false;
        this.actualizar_Datos();

        for(ArrayList <Cupo> cupos_por_dia: this.agenda_dias) {

            for(Cupo cupo: cupos_por_dia) {

                if (cupo.isDisponible()==true) {
                    booleano=true;
                    break;
                }
            }
        }
        return booleano;
    }

```

```java
    public void actualizar_Datos() {

        for(ArrayList <Cupo> Array_dia : this.agenda_dias) {
            Cupo.actualizarCupo(Array_dia);
        }
    }
```

El método tieneCupos(), en su lógica, lo primero que hará es actualizar los cupos del empleado desde el día actual hasta los próximos cinco días, mediante el método actualizarDatos() que habilita los cupos de los días posteriores, seguidamente lo que hará es recorrer el ArrayList de objetos de tipo cupo del empleado y verificar si alguno tiene disponibilidad con el metodo del la clase Cupo actualizarCupo().

```java
static public void actualizarCupo(ArrayList <Cupo> Array_dia){
    LocalDate fecha_Actual = LocalDate.now();

        //Se comprueba que la fecha sea anterior a la actual.
        if (Array_dia.get(index:0).getDia().isBefore(fecha_Actual)) {


            //Si el dia coincide con el actual, se habilian los cupos para hoy.
            if (fecha_Actual.getDayOfWeek().getValue() == Array_dia.get(index:0).getDia().getDayOfWeek().getValue())

                Array_dia.clear();

                Array_dia.add(new Cupo(fecha_Actual, horaInicio:"8:00",horaFin:"10:00", disponible:true));
                Array_dia.add(new Cupo(fecha_Actual, horaInicio:"10:00",horaFin:"12:00", disponible:true));
                Array_dia.add(new Cupo(fecha_Actual, horaInicio:"14:00",horaFin:"16:00", disponible:true));
                Array_dia.add(new Cupo(fecha_Actual, horaInicio:"16:00",horaFin:"18:00", disponible:true));

            }
```

```
76          else {
77
78              fecha_Actual = LocalDate.now();
79
80              int num_dia=Array_dia.get(index:0).getDia().getDayOfWeek().getValue();
81
82              boolean continuar = true;
83
84             while(continuar) {
85
86                 //Si el dia no coincide, buscamos entre los proximos.
87
88                  fecha_Actual = fecha_Actual.plusDays(daysToAdd:1); //Pasamos al dia siguiente.
89
90                 if (fecha_Actual.getDayOfWeek().getValue() == num_dia)  {
91
92                   Array_dia.clear();
93
94                   //Llenamos el dia por delante
95                   Array_dia.add(new Cupo(fecha_Actual, horaInicio:"8:00",horaFin:"10:00", disponible:true));
96                   Array_dia.add(new Cupo(fecha_Actual, horaInicio:"10:00",horaFin:"12:00", disponible:true));
97                   Array_dia.add(new Cupo(fecha_Actual, horaInicio:"14:00",horaFin:"16:00", disponible:true));
98                   Array_dia.add(new Cupo(fecha_Actual, horaInicio:"16:00",horaFin:"18:00", disponible:true));
99
100                  continuar = false;
```

El valor resultante, que es un boolean, lo tomará el método tieneEmpleado() para saber si agregar o no al empleado en el ArrayList de empleadosDisponibles que será retornado al final del método.

Si el ArrayList que se retorna a la clase main, está vacío, es porque en la sede no hay empleados con disponibilidad para atender citas, por lo cual se le informará que el proceso no podrá continuar por falta de disponibilidad. De no ser así, es porque existen empleados con disponibilidad para atender a las mascotas.

Hecho esto, el usuario tendrá la posibilidad de escoger el empleado de su preferencia entre los disponibles; posterior a esto podrá seleccionar el día en el que desea agendar la cita, cabe mencionar que las citas se agendan semanalmente, es decir desde el día actual, hasta los próximos cinco días, sin incluir el domingo. Después de seleccionar el día, al empleado seleccionado se le aplicará el método cupos_disponibles() el cual recibirá un entero correspondiente al día de la semana, siguiendo el orden, uno para lunes, dos para martes, tres para miércoles y consecutivamente.

```
693         //CUPOS QUE TIENE EL EMPLEADO PARA EL DIA SELECCIONADO
694         ArrayList <Cupo> cupos_disponibles = empleado_seleccionado.cupos_disponibles(num_dia);
```

```
144     public ArrayList<Cupo> cupos_disponibles(int i){
145
146         ArrayList<Cupo> cupos_disponibles= new ArrayList<>();
147
148         for(ArrayList <Cupo> Array_dia : this.agenda_dias) {
149
150             if (Array_dia.get(index:0).getDia().getDayOfWeek().getValue()==i) {
151
152                 for(Cupo cupo: Array_dia) {
153
154                     if(cupo.isDisponible()) {
155                     cupos_disponibles.add(cupo);
156                     }
157                 }
158                 break;
159             }
160         }
161
162         return cupos_disponibles;
163     }
```

El método retornará un ArrayList con los cupos que tengan disponibilidad en el día seleccionado por el usuario, si el ArrayList que retorna, está vacío, se le informará al cliente que no se puede continuar con el proceso agendar servicio a falta de disponibilidad de cupos por parte del empleado.

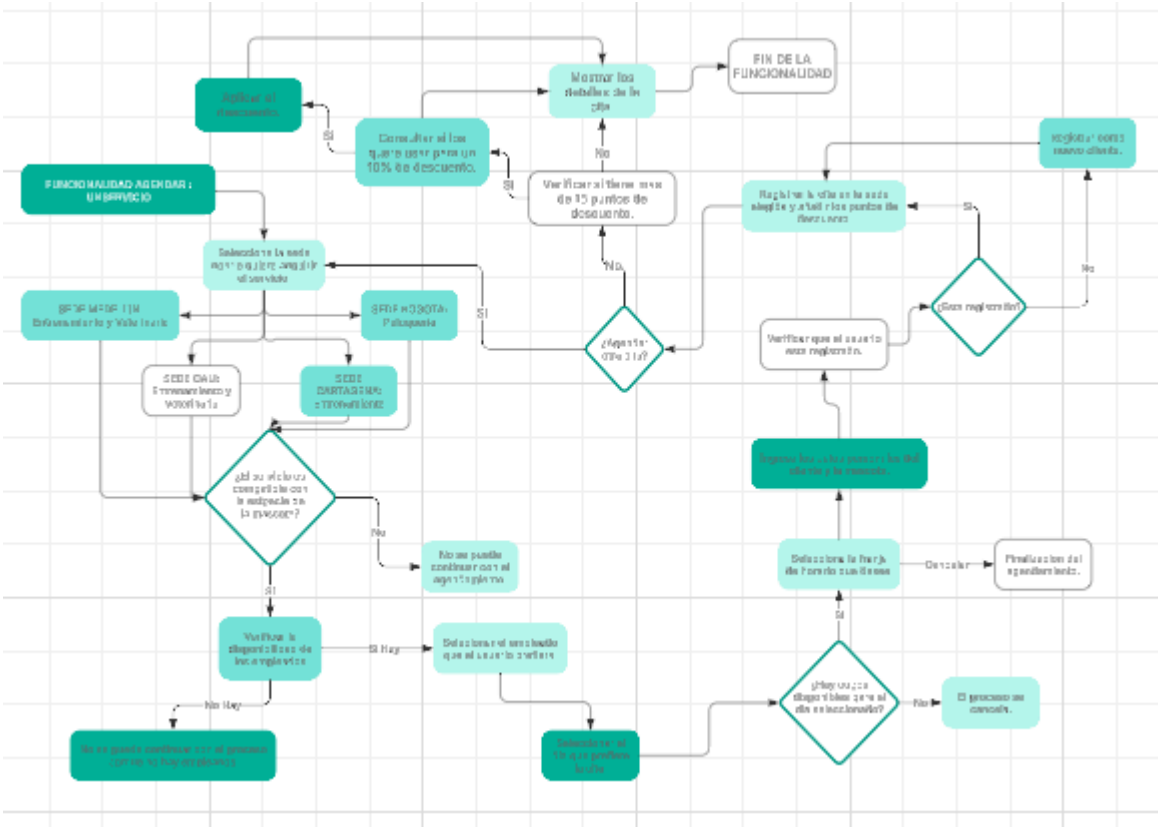
De otro modo, el usuario podrá seleccionar el cupo que sea de su grado en el horario ofertado. Para completar el agendamiento del servicio, se le pedirá al usuario que ingrese sus datos personales y los datos de la mascota para la que se desea agendar el servicio. Con los datos ingresados, se crearán los objetos de tipo Cliente y tipo Mascota correspondientemente; para así también crear el objeto Cita, el cual en su constructor recibirá los objetos de tipo Cliente, tipo Mascota, el empleado seleccionado y el cupo seleccionado, y dependiendo de cuál haya sido el servicio seleccionado, se le dará cierta cantidad de puntos de fidelidad al cliente, estos serán: para Entrenamiento 10 puntos, para Veterinaria 8 puntos y para Peluqueria 5 punto. Al usuario se le brindará la posibilidad de agendar otras citas adicionales a la actual.

Por último, se verifica si el cliente tiene más de 15 puntos, si es así, entonces se le preguntará si quiere usarlo para obtener un 10% de descuento en el costo de sus citas, o si desea seguir acumulando. Independiente de su elección por pantalla se le mostrarán los detalles de las citas agendadas.

Si todo sale bien al momento de generar una cita, se tendrá algo como esto

```
?? DETALLES DE LAS CITAS AGENDADAS ??
Cliente: mel
Mascota: m
Veterinario: Ana
Costo cita: 80000.0 pesos
Fecha de la cita: 29 enero 2025
Hora: 8:00 - 10:00
-------------------------------------


¡Cita agendada exitosamente!
```

DIAGRAMA DE INTERACCIÓN:



https://lucid.app/lucidchart/6a9620a0-baf6-4287-a115-479b6a095fe8/edit?viewport_loc=-684%2C39%2C3570%2C1407%2C0_0&invitationId=inv_50023d5e-46b7-499d-b750-55da22dc0b9d

- **FUNCIONALIDAD 3:** Tienda UNamascota

En el main, podemos llamar al método tienda(), cuyo funcionamiento consiste en crear un objeto de tipo Tienda y anexar a una sede. Una vez creado, el objeto de tipo Tienda comienza con una lista de objetos de tipo Producto (que tienen los atributos de nombre, tipo de animal, tipo de uso, precio y unidades). Los productos de esta lista son los que se ofertarán en la tienda.

```
public static void tienda() {



    //CREAMOS UN EMPLEADO PARA QUE ATIENDA LA TIENDA
    Empleado empliado = new Empleado(nombre:"Albert", edad:22, cedula:555, telefono:1323, direccion:"West Elm", Empleado.Especialidad.VENDEDOR);

    // CREACIÓN DE TIENDA
    //t1 = new Tienda(empliado, sede1);
    Tienda t1 = new Tienda(empliado,sede1);
```

```
public class Producto implements Serializable {
    //---> Atributos <---
    private static final long serialVersionUID = 1L;
    private String nombre; //Nombre del producto.
    private float precio; //Precio individual del producto.
    private String tipoAnimal; //Tipo de animal que puede usar el producto.
    private int cantidadUnidades; //Cantidad de unidades disponibles.
    private String tipoUso; //Uso al que va orientado el producto.

    //---> Constructores <---
    public Producto(String nombre, float precio, String tipoAnimal, String tipoUso, int cantidadUnidades){
        this(nombre, precio, tipoUso, cantidadUnidades); // Reutiliza el segundo constructor
        this.tipoAnimal = tipoAnimal; // Se inicializa correctamente aquí
    }
```

Antes de comenzar la interacción con el usuario, se corrobora que el objeto creado de tipo Tienda tenga un atributo de tipo Empleado asignado (Que se haya creado dentro del main para su uso como filtro o verificación), de ser el caso el código continúa con las primeras interacciones con el usuario, de lo contrario se mostrará que la tienda no tiene a nadie que pueda atender a la solicitud de la funcionalidad necesaria.

```
//CONSTRUCTORES//
public Tienda(Empleado empleado, CentroAdopcion centroAdopcion){
    Tienda.empleados.add(empleado); //TIENDA CON UN EMPLEADO
    this.centroAdopcion = centroAdopcion;

}
```

```
92              }
93              else {
94                  return "🚫Lo lamentamos pero el centro UNamascota no tiene empleados disponibles para atender su solicitud.🚫";
95              }
96          }
```

Después de este primer proceso, en el método tienda() de la clase main, se le pregunta al usuario: ¿Cómo desea que se le muestren los productos? El usuario tiene dos opciones: filtrar por el tipo de animal al que va dirigido el producto o mostrar todos los productos. Dependiendo de la respuesta, se evalúa en un condicional, y acto seguido se invoca al objeto

Tienda, que con su método inventario() o filtrar(), retorna un String con los productos que el usuario desea ver.

```java
public String inventario(){ //INVENTARIO
    if(empleados!=null){ // SI HAY UN EMPLEADO SE HACE INVENTARIO
        String resultado = ""; //STRING DONDE SE VA A GUARDAR EL TEXTO
        int indice = 1; // ÍNDICE QUE SE VA A MOSTRAR ENCIMA DEL PRODUCTO

        for (int i = 0; i<productos.size();i++) {
            resultado += indice+"."+productos.get(i).toString(); // RECORREMOS LA LISTA, CONCATENANDO EL ÍNDICE CON EL PRODUCTO EN STRING
            indice = indice+1;
        }
        return resultado; // SE DEVUELVE EL RESULTADO
    }
    else {
        return "🚫Lo lamentamos pero el centro UNamascota no tiene empleados disponibles para atender su solicitud.🚫";
    }
}
```

```java
public String filtrar(String tipo){ //FILTRAR
    if(empleados!=null){ // SI HAY POR LO MENOS UN EMPLEADO EN LA TIENDA, SE HACE EL FILTRADO DE PRODUCTOS
        String resultado = ""; //SE CREA UN STRING DONDE SE VA A ALMACENAR LOS PRODUCTOS CON UN ÍNDICE
        int indice = 0;
        for (int i = 0; i<productos.size();i++) { // SE RECORRE LA LISTA
            indice = indice+1; //ÍNDICE QUE SE MUESTRA POR UNCIMA DEL PRODUCTO
            if (productos.get(i).getTipoAnimal().equals(tipo) || productos.get(i).getTipoAnimal().equals(anObject:"Uso general")) { // SI LOS PRODUCTOS SON DE USO GENERAL, O DE
                resultado += indice+"."+productos.get(i).toString(); // SE ANEXAN LOS PRODUCTOS AL STRING QUE SE VA A RETORNAR
            }
        }
        return resultado;
    }
    else {
        return "🚫Lo lamentamos pero el centro UNamascota no tiene empleados disponibles para atender su solicitud.🚫";
    }
}
```

```java
@Override
public String toString(){
    return "\nProducto: "+getNombre()+"\n"+"Precio: "+getPrecio()+"\n"+"Destinado a: "+getTipoAnimal()+"\n"+"Tipo del Producto: "+getTipoUso()+"\n"+ "Cantidad unidades: "+getCantidadUnidades()+"\n";
}
}
```

Una vez se muestran los productos al usuario, se le pide un índice que indica la posición en la lista del producto que desea comprar. Acto seguido, se le solicita la cantidad de unidades que quiere adquirir, y luego se vuelve a llamar a nuestro objeto Tienda. Este, con su método compra(), se encarga de restar la cantidad de unidades del producto solicitado y, posteriormente, retorna una pequeña factura con el monto a pagar.

```
Productos disponibles:
1.🔴
Producto: Croquetas de Pollo para perro
Precio: 50000.0
Destinado a: Perro
Tipo del Producto: Alimentación
Cantidad unidades: 100
```

```java
public String compra(int indice, int unidades, Cliente cliente){

    if (empleados!=null){ // SE COMPRUEBA SI HAY UN EMPLEADO PARA LA OPERACION DE LA TIENDA

        indice -= 1; //CONVERTIMOS EL NÚMERO A INDICE

        if (indice>=productos.size() || indice<0) { //COMPROBAMOS SI EL INDICE ES CORRECTO
            return "🚫¡Oh no!, verifica que el índice del producto sea el indicado, el centro UNamascota desconoce ese índice.🚫";
        }
        int cantidad = productos.get(indice).getCantidadUnidades(); //GUARDAMOS LA CANTIDAD DE UNIDADES DEL PRODUCTO A COMPRAR

        if (cantidad!=0 && cantidad>=unidades){ //SI LA CANTIDAD INGRESADA ESTÁ DENTRO DEL RANGO INICIA LA OPERACION DE LA TIENDA

            cantidad-=unidades; // SE RESTA LAS UNIDADES QUE SE COMPRARON
            productos.get(indice).setCantidadUnidades(cantidad); // SE CAMBIA LAS UNIDADES RESTANTES EN EL PRODUCTO
            String nombre = productos.get(indice).getNombre();
            String tipo = productos.get(indice).getTipoAnimal();
            if (productos.get(indice).getCantidadUnidades()==0) {
                productos.remove(indice); // SI LA CANTIDAD LLEGA A 0, SE ELIMINA EL PRODUCTO

            }

            cliente = CentroAdopcion.esCliente(cliente); // SE COMPRUEBA SI EL CLIENTE EXISTE
            float precio = productos.get(indice).getPrecio(); // SE GUARDA EL PRECIO PARA GENERAR DESPUES FACTURA
            int puntos = cliente.getPuntos(); // SE TOMAN LOS PUNTOS

            if (puntos>=15) { // SI TIENE MÁS DE 15 PUNTOS ACOMULADOS, SE LE HACE UN DESCUENTO AUTOMÁTICAMENTE
                cliente.disminuir_Puntos(puntos); // SE RESTAN LOS 15 PUNTOS QUE SE USARON COMO DESCUENTO
                precio -= precio*unidades*0.1; // SE CALCULA EL PRECIO CON EL 10% DE DESCUENTO
                return "----------------------------------------\n"+"Has comprado "+unidades+" unidades de: "+nombre+" - Dirigido a: "+tipo+"\n"+"Total a pagar: "+precio+" $"+"\n"+"Han sido descontados 15
            }else { // RETORNA UNICAMENTE EL PRECIO PORQUE NO TIENE LOS PUNTOS SUFICIENTES
                return "----------------------------------------\n"+"Has comprado "+unidades+" unidades de: "+nombre+" - Dirigido a: "+tipo+"\n"+"Total a pagar: "+precio*unidades+" $"+"\n"+"No cuentas con l
            }
        }
        else{
            return "🚫¡Oh no!, verifica la cantidad de unidades disponibles en la tienda y vuelve a intentarlo.🚫";
        }
    }else{
        return "🚫Lo lamentamos pero el centro UNamascota no tiene empleados disponibles para atender su solicitud.🚫";
    }
}
```

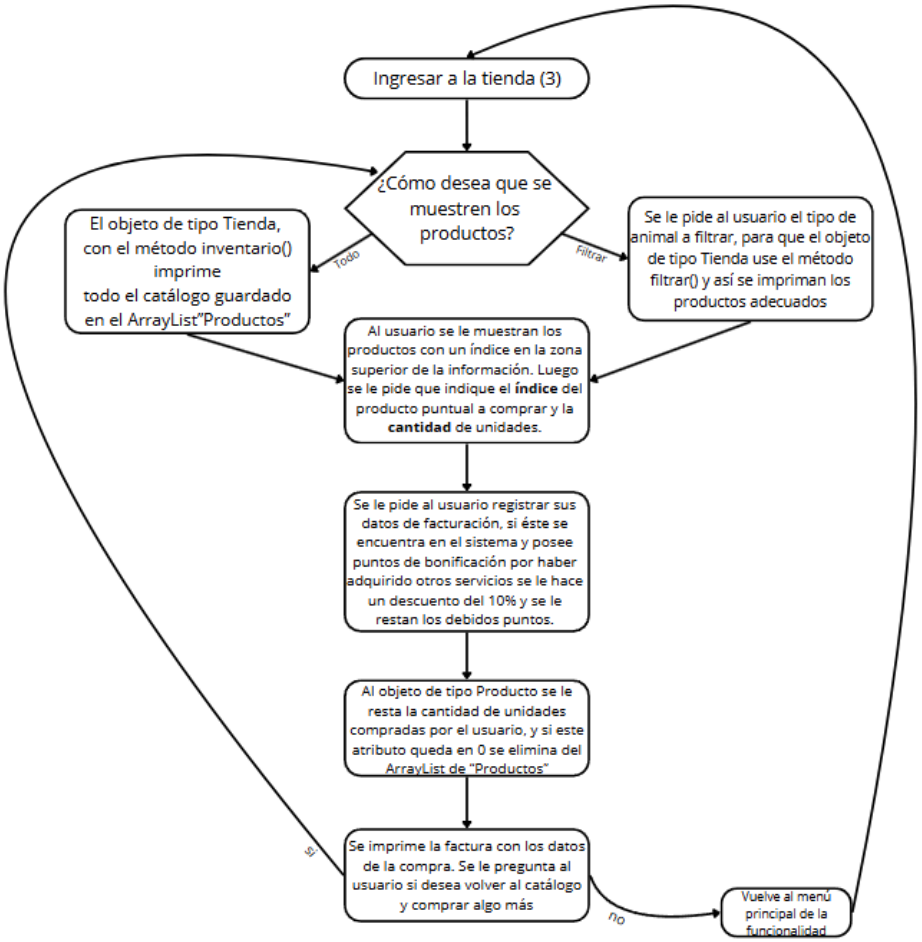
Si la tienda detecta que, al momento de hacer la compra, el usuario está registrado (con el

método compra() que recibe también un objeto de tipo Cliente) o ha sido comprador de alguno de los servicios, y además cuenta con más de 15 puntos de bonificación, recibirá un descuento del 10% en su compra.

Finalmente, es importante tener en cuenta que la tienda tiene su propia lista de empleados, y necesita que al menos un objeto de tipo Empleado esté en esta lista para poder realizar las acciones de mostrar productos (métodos inventario() o filtrar()).

```
if (empleados!=null){ // SE COMPRUEBA SI HAY UN EMPLEADO PARA LA OPERACION DE LA TIENDA
```

```
}else{
    return "🚫Lo lamentamos pero el centro UNamascota no tiene empleados disponibles para atender su solicitud.🚫";
}
```



- FUNCIONALIDAD 4: Memorial para mascotas:

En el **main** podemos llamar al método **gestionarMemorial(),** el cual crea un objeto de tipo Memorial.
La funcionalidad primero pide los datos del usuario en caso de querer crear un memorial para su mascota difunta. Luego se despliega un menú con las opciones:

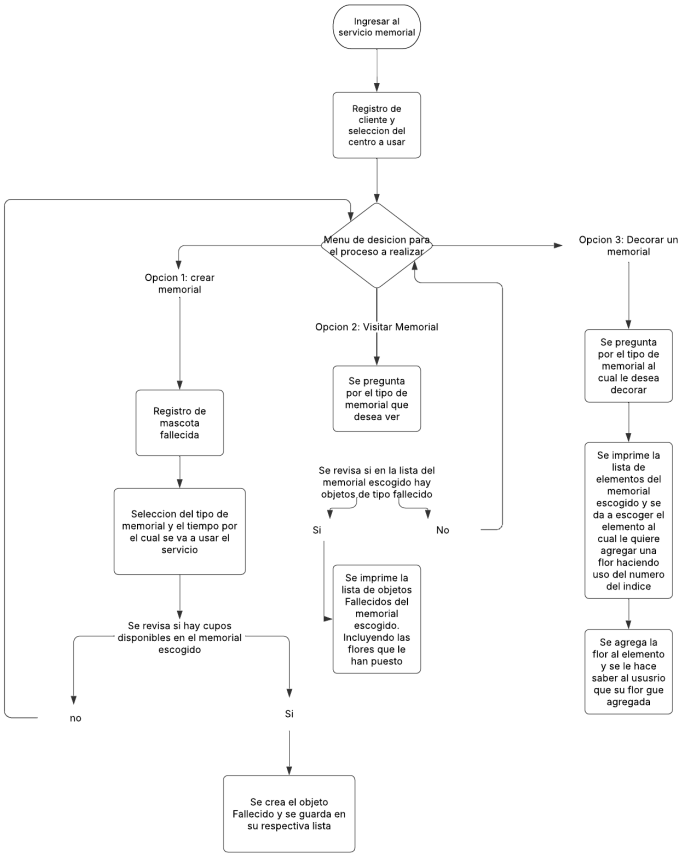
1. Añadir memorial: Se encarga de crear un objeto de tipo **Fallecido**, que lleva en los atributos los detalles que se mostrarán en el memorial creado, como: nombre de la mascota, fecha de fallecimiento y un mensaje dejado por su dueño.

Luego de haber creado el objeto, se guarda en una lista dentro del objeto **Memorial** con el respectivo tipo de memorial que se desee crear para la difunta mascota (Cenizas, Osario, Sepulcro o Plantar un árbol). El **main** luego preguntará por el tiempo durante el cual se desea emplear el servicio memorial, teniendo la opción de usar el memorial de por vida o por un tiempo limitado (se puede escoger años en múltiplos de 5). Independientemente de la acción, se imprimirá en pantalla el monto a pagar por el uso del memorial. En caso de que el proceso haya salido de manera correcta, el **main** imprimirá un mensaje informando al dueño que el proceso fue exitoso. De lo contrario (si el memorial escogido no tiene cupos), se le hará saber al dueño que no hay cupos suficientes y el proceso de creación del memorial terminará.

2. Ver memorial: El **main** se encargará de preguntar qué tipo de memorial desea visitar (Cenizas, Osario, Sepulcro o los árboles plantados) perteneciente al objeto Memorial. Dependiendo del tipo de memorial que se desee visitar, se hace uso del método **obtenerFallecidosPorTipo()**, que recibe un parámetro tipo y selecciona la lista respectiva al tipo de memorial que se seleccionó (las listas hacen parte del objeto **Memorial: ArrayList<Fallecido> tipo**). Luego, se hace uso del método **visitaMemorial()**, el cual imprime el índice (+1) seguido de los objetos Fallecido que forman parte de la lista seleccionada, incluyendo las decoraciones que se le han dejado.

3. Decorar memorial: El **main** preguntará el tipo de memorial que se quiere decorar (Cenizas, Osario, Sepulcro o los árboles plantados). Luego, se hace uso del método **obtenerFallecidosPorTipo()**, que imprimirá los elementos de la lista del tipo seleccionado indicados por un índice. Dicho índice puede ser usado por el usuario para escoger el objeto de tipo Fallecido al cual quiere dejar flores. El método **ponerFlor()** se encarga de guardar en un **ArrayList<>** todas las flores que se le han dejado al objeto de tipo **Fallecido**.

Diagrama de flujo funcionalidad 4:



https://lucid.app/lucidchart/8d17bf81-0bb4-4795-9c4b-e65cfbdce62c/edit?viewport_loc=109%2C939%2C2244%2C1053%2C0_0&invitationId=inv_89fecb3b-8155-4d2b-8cf9-a4a5d2c9b66b

- FUNCIONALIDAD 5: Planificación de Dietas para Mascotas:

En el main, se llama al método planificacionDieta(), que tiene como objetivo registrar a un cliente y su mascota para generar un plan alimenticio personalizado.

Fases de la funcionalidad:

1. Registro del Cliente y la Mascota

Una vez iniciado el método, el sistema solicita los datos del usuario, incluyendo: Nombre completo, Edad y Número de identificación. Despues, se solicitan los datos de la mascota, su nombre, especie, edad, sexo, tamaño y peso (kilos). Solo se aceptaran perros y gatos, los tamaños consisten en Miniatura, Pequeño, Mediano, Grande El sistema valida que la especie ingresada sea válida y que los datos sean coherentes antes de continuar.

2. Cálculo del Peso Ideal

Después del registro, se crea un objeto Mascota con los datos introducidos, se invoca el método

calcularPesoIdeal(), el cual evalúa el peso ideal de la mascota utilizando la fórmula:

```
//Crear un objeto Mascota con los datos que ingresó el usuario
Mascota mascota = new Mascota(nombre, tipo, edad, sexo, EstadoSalud.SANO, tamano, peso);
//Crea el objeto dieta asociado a la mascota
Dieta dieta = new Dieta(mascota);
//utiliza las formulas aritmeticas de los metodos calcularPesoIdeal() y planDieta() para calcular el porcentaje de nutrientes que debe
//consumir la mascota, dependiendo de su debe subir o bajar de peso.
dieta.calcularPesoIdeal();
dieta.planDieta();
```

$$Peso\ ideal\ = \frac{tamaño \times 10}{edad + 1} + \frac{peso\ actual}{2}$$

El método comparacionPeso() compara su peso ideal con el peso actual y determinará si la mascota debe subir o bajar de peso.

```
public void calcularPesoIdeal(){
     int tamano = mascota.getTamano();
    int edad = mascota.getEdad();
    double pesoActual = mascota.getPeso();
    this.pesoIdeal = ((tamano * 10.0) / (edad + 1)) + (pesoActual / 2);
}

public int comparacionPeso() {
    if (mascota.getPeso() == this.pesoIdeal) {
        return 1; // Esta en su peso ideal
    } else if (mascota.getPeso() < this.pesoIdeal) {
        return 2; // Necesita subir de peso
    } else {
        return 3; // Necesita bajar de peso
    }
}
```

3. Planificación de la Dieta

El método planDieta() determinará la cantidad diaria de gramos de alimento que necesita la mascota utilizando la fórmula:

$$Gramos\ diarios\ =\ Peso\ Ideal\ \times\ Tamaño\ \times\ 10$$

y después utilizando el método comparacionPeso() determinará la distribución de proporciones de nutrientes:
Si necesita subir de peso: 30% grasas, 40% proteínas, 30% carbohidratos.

Si necesita bajar de peso: 15% grasas, 50% proteínas, 35% carbohidratos.

Si está en su peso ideal: 20% grasas, 30% proteínas, 50% carbohidratos.

El sistema calcula la cantidad diaria de alimento necesaria y la distribuye en gramos de proteínas, carbohidratos y grasas.

```
public void planDieta() {
    this.gramosDiarios = this.pesoIdeal * mascota.getTamano() * 10;
    switch (comparacionPeso()) {
        case 1: //Esta en su peso ideal
            this.grasas = this.gramosDiarios * 0.20;
            this.proteinas = this.gramosDiarios * 0.30;
            this.carbohidratos = this.gramosDiarios * 0.50;
            break;
        case 2: //Subir de peso
            this.grasas = this.gramosDiarios * 0.30;
            this.proteinas = this.gramosDiarios * 0.40;
            this.carbohidratos = this.gramosDiarios * 0.30;
            break;
        case 3: //Bajar de peso
            this.grasas = this.gramosDiarios * 0.15;
            this.proteinas = this.gramosDiarios * 0.50;
            this.carbohidratos = this.gramosDiarios * 0.35;
            break;
    }
}
```

4. Generación del Menú Nutricional

El método toString() crea un menú sugerido basado en los productos disponibles en la tienda, y un pequeño resumen del estado físico de la mascota. Además, el método menu() creará un archivo .txt que guardará lo que se imprima con toString().

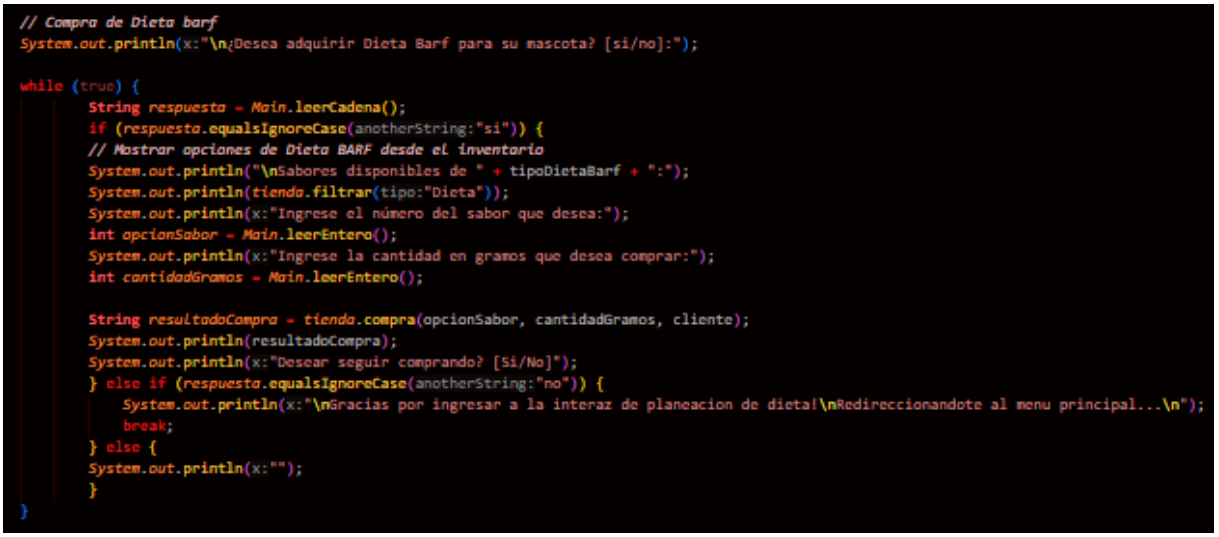
```
    @Override
public String toString() {
    // Determinar el estado de peso
    String estadoPeso;
    switch (comparacionPeso()) {
        case 1:
            estadoPeso = "Está en su peso ideal.";
            break;
        case 2:
            estadoPeso = "Debe subir de peso.";
            break;
        case 3:
            estadoPeso = "Debe bajar de peso.";
            break;
        default:
            estadoPeso = "Estado no definido.";
    }

    return "Nombre de la mascota: " + mascota.getNombre() + "\n" +
           "Peso Actual: " + mascota.getPeso() + " kg\n" +
           "Edad: " + mascota.getEdad() + " años\n" +
           "Tamaño: " + mascota.getTamanoString() + "\n" +
           "Peso ideal: " + Math.round(this.pesoIdeal * 100) / 100d + " kg\n" +
           "Cantidad de Gramos de alimento diarios: " + Math.round(this.gramosDiarios * 100) / 100d + " g\n" +
           estadoPeso + "\n\n" +
           "Distribución en porcentajes de nutrientes:\n" +
           " Proteínas: " + ((this.proteinas / this.gramosDiarios) * 100) + "%\n" +
           " Grasas: " + ((this.grasas / this.gramosDiarios) * 100) + "%\n" +
           " Carbohidratos: " + ((this.carbohidratos / this.gramosDiarios) * 100) + "%\n\n" +
           "Distribución en gramos de nutrientes:\n" +
           " Proteínas: " + Math.round(this.proteinas * 100) / 100d + " g\n" +
           " Grasas: " + Math.round(this.grasas * 100) / 100d + " g\n" +
           " Carbohidratos: " + Math.round(this.carbohidratos * 100) / 100d + " g";
}
```

5. Integración con la Tienda Virtual

Después de la generación de la dieta, el usuario tiene la opción de adquirir Dieta Barf para su mascota en la tienda virtual. El sistema determinará si se necesita Dieta Barf para gato o perro, y presentará el catálogo de alimentos disponibles a granel.

```
// Determinar el tipo de Dieta barf (perro o gato)
String tipoDietaBarf = tipo.equalsIgnoreCase(anotherString:"Perro") ? "Dieta Barf para Perros" : "Dieta Barf para Gatos";

// Creacion de la tienda y de los productos dieta barf.
Tienda tienda = new Tienda(new Empleado(nombre:"Albert", edad:22, cedula:555, telefono:1323, direccion:"West Elm", Empleado.Especialidad.VENDEDOR));
Producto[] productosBarf = {
    new Producto("Dieta Barf Alto en Proteinas para " + tipo + " (Gramo)" , precio:45f, tipoAnimal:"Dieta", "Alimento para " + tipo , cantidadUnidades:1000),
    new Producto("Dieta Barf Alto en Grasas para " + tipo + " (Gramo)" , precio:45f, tipoAnimal:"Dieta", "Alimento para " + tipo, cantidadUnidades:1000),
    new Producto("Dieta Barf Alto en Carbohidratos para " + tipo + " (Gramo)" , precio:45, tipoAnimal:"Dieta", "Alimento para " + tipo , cantidadUnidades:1000)
};

for (Producto producto : productosBarf) {
    Tienda.agregarProducto(producto);
}
```

El usuario elige un producto y la cantidad en gramos que desea comprar y se invoca el método compra() de la clase Tienda, el cual verifica el inventario y procesa la transacción. Si el usuario cuenta con más de 15 puntos de bonificación, recibe un 10% de descuento en su compra.

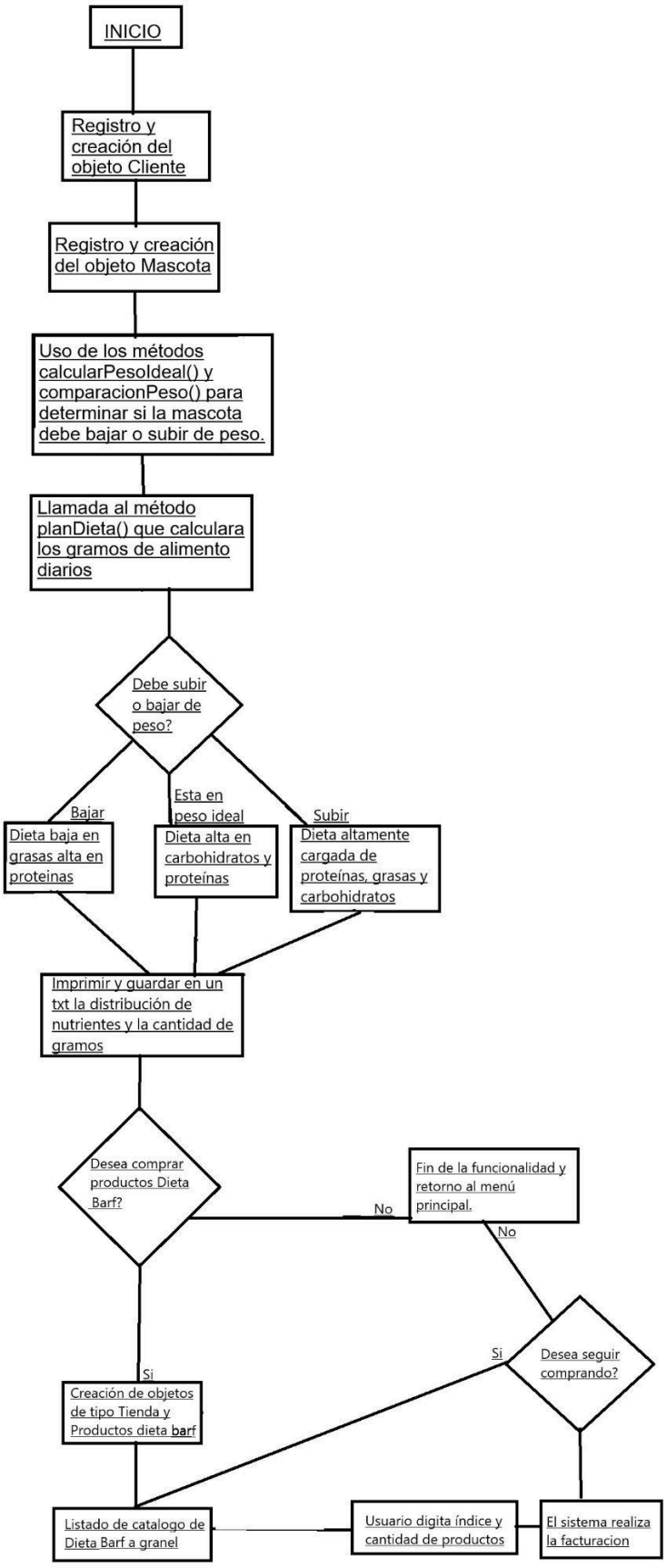


```java
// Compra de Dieta barf
System.out.println(x:"\n¿Desea adquirir Dieta Barf para su mascota? [si/no]:");

while (true) {
        String respuesta = Main.leerCadena();
        if (respuesta.equalsIgnoreCase(anotherString:"si")) {
        // Mostrar opciones de Dieta BARF desde el inventario
        System.out.println("\nSabores disponibles de " + tipoDietaBarf + ":");
        System.out.println(tienda.filtrar(tipo:"Dieta"));
        System.out.println(x:"Ingrese el número del sabor que desea:");
        int opcionSabor = Main.leerEntero();
        System.out.println(x:"Ingrese la cantidad en gramos que desea comprar:");
        int cantidadGramos = Main.leerEntero();

        String resultadoCompra = tienda.compra(opcionSabor, cantidadGramos, cliente);
        System.out.println(resultadoCompra);
        System.out.println(x:"Desear seguir comprando? [Si/No]");
        } else if (respuesta.equalsIgnoreCase(anotherString:"no")) {
            System.out.println(x:"\nGracias por ingresar a la interaz de planeacion de dieta!\nRedireccionandote al menu principal...\n");
            break;
        } else {
        System.out.println(x:"");
        }
}
```

6. Finalización y Redirección

Al finalizar, el usuario puede optar por continuar comprando productos o salir de la funcionalidad. Si decide salir, el sistema muestra un mensaje de confirmación y lo redirige al menú principal.

Diagrama de Interacción:

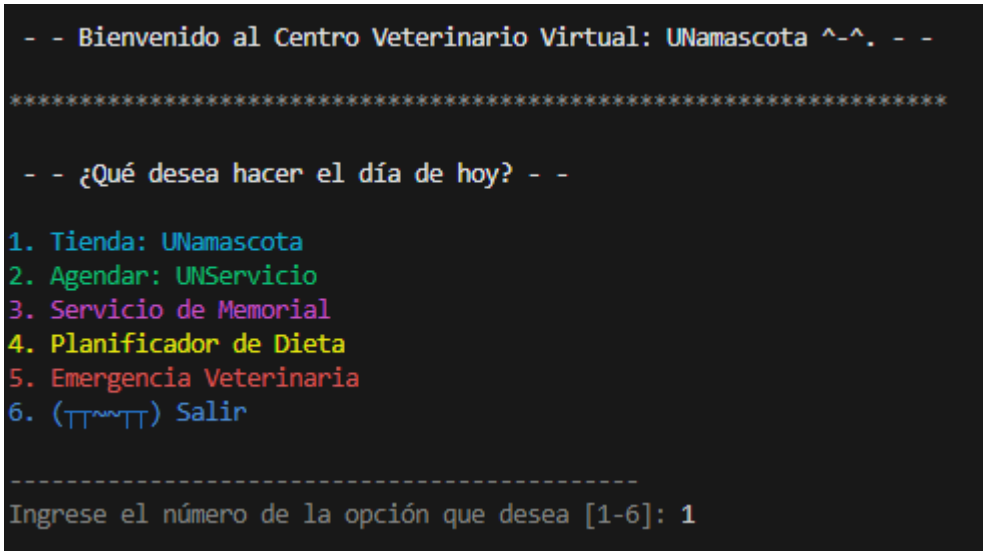


6. Manual de usuario.

En primer lugar recibirás un archivo con finalización (.zip) Deberás abrirlo, descomprimirlo y posteriormente tendrás que ejecutar el archivo que finalice en (.bat) Cuando ejecutes este archivo te aparecerá un menú compuesto por cinco opciones, en pantalla se tendrá que ver así:

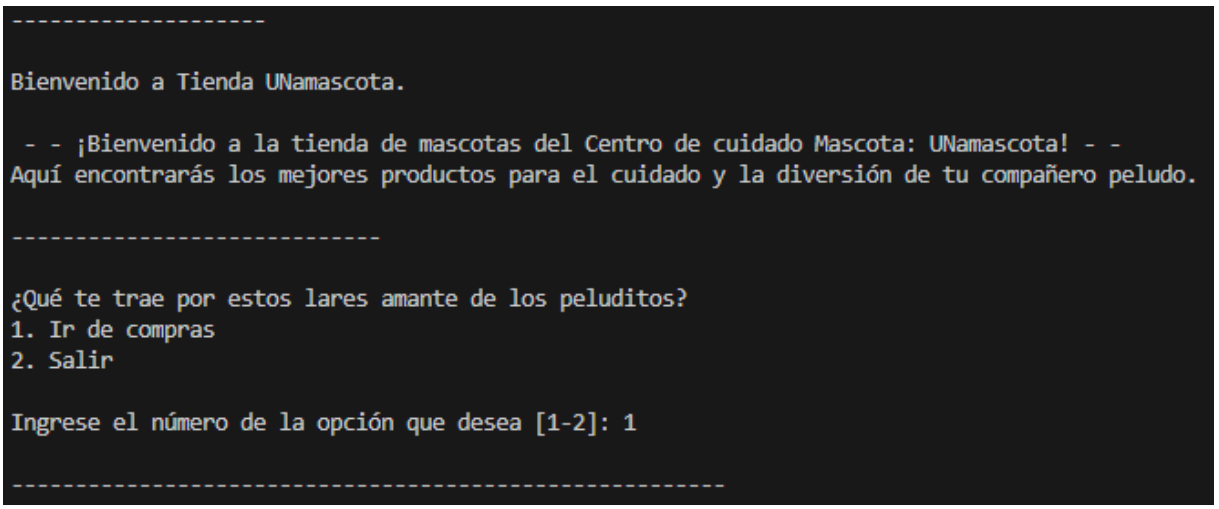
En este momento tendrás que ingresar por tu teclado, un número entre 1 y 6, de acuerdo a la opción que quieras acceder; por ejemplo, si deseas agendar una cita, tendrás que pulsar el número 2 en tu teclado. Cabe recalcar que para avanzar se necesita que oprima la barra enter de su computador. Si no se le solicita ningún valor oprima enter solamente.

De acuerdo a lo que selecciones, tendrás diferentes resultados:



Si seleccionaste el número (1): Tienda UNamascota

Inicialmente se te preguntará qué deseas hacer al entrar a la tienda: 1. Ir de compras - 2. Salir



Si seleccionas la opción 1. Ir de compras: Aquí te preguntaremos cómo deseas que se te muestren los productos que tenemos disponibles, entre las opciones que te mostramos está: 1. Mostrar todo (Mostrar todo el catálogo de productos que tenemos) o 2. Filtrar por tipo (Si deseas buscar un producto específicamente para tu tipo de mascota)

```
------------------------------------------------------

¿Cómo desea que se le muestren los productos?
1. Mostrar todo
2. Filtrar por tipo

Ingrese el número de la opción que desea [1-2]:

------------------------------------------------------
```

1. Mostrar todo: Se imprimirá por consola todos los productos disponibles en la tienda, sobre ellos se mostrarán los índices al que pertenece cada producto, para que de tal forma, luego de saber el producto que deseas comprar, puedas indicarle al programa el número específico del producto. Una vez ingresado el índice del producto se te preguntará la cantidad de unidades que deseas comprar y una vez realizado eso se te llevará a un apartado de datos de compra para poder proceder con la impresión de tu recibo de pago. Te sugerimos que revises bien la información que nos provees para lograr que todo el proceso se lleve a cabo correctamente.

```
------------------------------------------------------
1

Productos disponibles:
1.
Producto: Croquetas de Pollo para perro
Precio: 50000.0
Destinado a: Perro 🔴
Tipo del Producto: Alimentación
Cantidad unidades: 100
2.
Producto: Croquetas de Pollo para gato
Precio: 45000.0
Destinado a: Gato 🔴
Tipo del Producto: Alimentación
Cantidad unidades: 50
3.
Producto: Chips de zanahoria para Conejo
Precio: 15000.0
Destinado a: Conejo🔴
Tipo del Producto: Alimentacion
Cantidad unidades: 30
4.
Producto: Alpiste
Precio: 30000.0
Destinado a: Ave🔴
Tipo del Producto: Alimento
Cantidad unidades: 30
```

```
Sus datos serán tomados para registrar la compra.
Ingrese su cédula: 1018232853
Ingrese su edad: 18
Ingrese su nombre: Tomas Ospina

-----------------------------------------
Muchas gracias por tu compra, adquiriste: Croquetas de Pollo para perro - Dirigido a: Perro
Total a pagar: 50000.0 $
No cuentas con los suficientes puntos como para un descuento
-----------------------------------------
```

Una vez llenados los datos te entregaremos tu recibo con la información de tu compra, el monto a pagar y si aplicas a un descuento por ser un cliente de las otras facciones de la veterinaria, en tal caso lo informaremos en tu recibo de pago.

```
----------------------------------------
Muchas gracias por tu compra, adquiriste: Croquetas de Pollo para perro - Dirigido a: Perro
Total a pagar: 50000.0 $
No cuentas con los suficientes puntos como para un descuento
----------------------------------------
```

Concluído eso te preguntaremos si deseas retornar al catálogo, en caso de hacerlo se te pedirá nuevamente como deseas que se te muestren tus productos.

```
----------------------------------------

¿Desea volver al catálogo? [si/no]:
```

En caso de no hacerlo te regresaremos al primer menú de la tienda para preguntar nuevamente si vas a comprar o salir de la funcionalidad.

2. Filtrar por tipo: Te preguntaremos por cuál tipo de mascotas (Disponibles en la tienda) te gustaría filtrar tu búsqueda de productos, por ejemplo en caso de que desees productos para tu perro escribirás “perros/perro” en tu consola para que el programa te muestre los productos disponibles dirigidos a tu mascota en específico.

```
----------------------------------------------------
2

¿Por qué tipo de Mascota te gustaría ver? [perros, gatos, aves o conejos]: conejos

Productos disponibles:

3.
Producto: Chips de zanahoria para Conejo
Precio: 15000.0
Destinado a: Conejo
Tipo del Producto: Alimentacion
Cantidad unidades: 30
7.
Producto: Cama para conejo
Precio: 80000.0
Destinado a: Conejo
Tipo del Producto: Hogar
Cantidad unidades: 10
```

Una vez hecho esto se te pedirá normalmente el índice del producto que deseas comprar, las unidades que deseas llevar y pasaremos a la sección donde te tomaremos tus datos para la generación de tu factura digital con la información de compra. Nuevamente, si tienes puntos acumulados por haber adquirido servicios de otras facciones de la veterinaria se te descontarán y se te hará un descuento del 10% en tu total final a pagar.

Concluído eso te preguntaremos si deseas retornar al catálogo, en caso de hacerlo se te pedirá nuevamente como deseas que se te muestren tus productos.

En caso de no hacerlo te regresaremos al primer menú de la tienda para preguntar nuevamente si vas a comprar o salir de la funcionalidad.

Si seleccionaste la opción 2. Salir: Te retornaremos al menú principal de la veterinaria para acceder al resto de funcionalidades.

Si seleccionaste el número (2):

Al comenzar, te preguntaremos a qué sede quieres acudir, tenemos las sedes Medellín, Bogotá, Cali y Cartagena, para ello, tendrás que ingresar un número entre 1 y 4.

Nuestras sedes pueden contar con los servicios de Entrenamiento, Veterinaria y Peluquería. Los servicios disponibles varían según la sede que selecciones.

Si seleccionaste SEDE MEDELLIN (1):

Se mostrará los servicios disponibles de la sede Medellín, la cual dispone del servicio de Entrenamiento y Veterinaria. Tendrás que ingresar un número entre 1 y 2 para agendar un servicio. Si deseas cancelar ingresa el numero 3 y volveras al menu de inicio.

Si seleccionaste SEDE BOGOTA (2):

Se mostrará los servicios disponibles de la sede Bogota, la cual dispone del servicio de Peluqueria. Tendrás que ingresar un número entre 1 para agendar el servicio. Si deseas cancelar ingresa el numero 3 y volveras al menu de inicio.

Si seleccionaste SEDE CALI (3):

Se mostrará los servicios disponibles de la sede Cali, la cual dispone del servicio de Entrenamiento y Veterinaria. Tendrás que ingresar un número entre 1 y 2 para agendar un servicio. Si deseas cancelar ingresa el numero 3 y volveras al menu de inicio.

Si seleccionaste SEDE CARTAGENA(4):

Se mostrará los servicios disponibles de la sede Cartagena, la cual dispone del servicio de Entrenamiento. Tendrás que ingresar un número entre 1 para agendar el servicio. Si deseas cancelar ingresa el numero 3 y volveras al menu de inicio.

De Acuerdo al servicio seleccionado se tendrán diferentes opciones:

Si seleccionaste el servicio de ENTRENAMIENTO:

Te informaremos el costo del servicio; ten presente que este servicio sólo se presta para perros y gatos; si no es de este tipo, no podremos prestarte este servicio.

A continuación podrás elegir entre uno de nuestros empleados, seleccionado el número que aparece antes de su nombre; ahora elige el día en el que deseas agendar el servicio, disponemos de lunes a sábado, para seleccionar el día, escribe el número que aparece antes del día que quieres seleccionar.

Ahora selecciona la hora para la que deseas agendar tu cita, de nuevo, escribe el número que aparece antes de tu elección.

En este punto, deberías poder ver un texto como:

Ahora ingresa tus datos y los datos de tu mascota, es decir, para la cual deseas agendar el servicio.

Se mostrará si tu cita fue agendada correctamente, si deseas agendar otro servicio, podrás escribir si y te devolveremos al menú de las sedes.

Al confirmar tu cita si registraste todo bien, tu pantalla debería verse similar a esta:

Si seleccionaste el servicio VETERINARIA:

Te informaremos el costo del servicio; ten presente que este servicio sólo se presta para perros, gatos, conejos y aves; si no es de este tipo, no podremos prestarte este servicio.

Podrás elegir entre uno de nuestros empleados, seleccionado el número que aparece antes de su nombre; ahora elige el día en el que deseas agendar el servicio, disponemos de lunes a sábado, para seleccionar el día, escribe el número que aparece antes del día que quieres seleccionar.

Si ingresas todo correctamente deberías ver un resultado similar:

Ahora selecciona la hora para la que deseas agendar tu cita, de nuevo, escribe el número que aparece antes de tu elección.

Ahora ingresa los datos de tu mascota, es decir, para la cual deseas agendar el servicio. Selecciona que tipo de animal es tu mascota, si es un perro, escribe 1; si es un gato, escribe 2; si es conejo escribe 3; finalmente si es un ave escribe 4.

Te informaremos si tu cita fue agendada correctamente, si deseas agendar otro servicio, podrás escribir si y te devolveremos al menú de las sedes.

Si tu cita fue agenda exitosamente, tendrás que ver el mensaje que observamos:

Si seleccionaste el servicio de PELUQUERIA:

Te informaremos el costo del servicio; ten presente que este servicio sólo se presta para perros y gatos; si no es de este tipo, no podremos prestarte este servicio.

A continuación podrás elegir entre uno de nuestros empleados, seleccionado el número que aparece antes de su nombre; ahora elige el día en el que deseas agendar el servicio, disponemos de lunes a sábado, para seleccionar el día, escribe el número que aparece antes del día que quieres seleccionar.

Ahora selecciona la hora para la que deseas agendar tu cita, de nuevo, escribe el número que aparece antes de tu elección.

En este punto, deberías poder ver un texto como:

Ahora ingresa tus datos y los datos de tu mascota, es decir, para la cual deseas agendar el servicio.

Se mostrará si tu cita fue agendada correctamente, si deseas agendar otro servicio, podrás escribir si y te devolveremos al menú de las sedes.

Al confirmar tu cita si registraste todo bien, tu pantalla debería verse similar a esta:

Finalmente, si ya terminaste de agendar tus servicios, podrás ingresar, no, y serás devuelto al menú principal.

Si seleccionaste el número (4) Planeación de dieta:

Entrarás al sistema de planificación de dieta. Esta funcionalidad está diseñada para ayudar a los dueños de mascotas a llevar un mejor control de la alimentación de sus mascotas para asegurarles una vida plena y saludable.

Registro:

1. Ingresa tu nombre completo.
2. Introduce tu edad en números enteros.
3. Proporciona tu número de identificación.
4. Introduce el nombre de tu mascota.
5. Especifica la especie (solo perros y gatos son permitidos).

6. Indica la edad de la mascota en números enteros.
7. Introduce el sexo de tu mascota (M/F).

8. Indica el tamaño de tu mascota

    1-Miniatura
    2-Pequeño
    3-Mediano
    4-Grande

9. Introduce el peso de tu mascota en kilogramos.

Resumen del estado físico de tu mascota.

El sistema te indicará de nuevo el peso actual de tu mascota, calculará el peso ideal de tu mascota automáticamente y te informará si la mascota necesita subir, bajar o mantener su peso.

Generación del Plan de Dieta

- Se generará una distribución diaria de alimentos basada en proteínas, carbohidratos y grasas.
- Se te mostrará la cantidad exacta de comida (en gramos) que debe consumir tu mascota diariamente.

Visualización del Menú Nutricional

- La aplicación mostrará un menú con la distribución en porcentajes y en gramos de los alimentos recomendados.
- Se generará un archivo .txt con la información nutricional de la dieta.
- Para abrir el archivo .txt, ve a la carpeta baseDatos, y ahí se guardará el archivo con el nombre de tu mascota.

Compra de Productos en la Tienda Virtual

El sistema te preguntará si deseas comprar productos dietéticos para tu mascota. Si indicas que no, se finalizará la funcionalidad. Si indicas que si:

1. El sistema listará los productos Dieta Barf disponibles por gramos.
2. Digita el índice del producto que desees comprar.
3. Digita la cantidad de gramos que desees comprar..
4. Si tienes más de 15 puntos de bonificación, recibirás un 10% de descuento en tu compra.
5. El sistema facturará tu compra y te preguntará si deseas seguir comprando o finalizar.

Finalización

Si decides salir, la aplicación te redirigirá al menú principal.El plan de dieta de tu mascota seguirá guardado en un archivo de texto de la carpeta baseDatos.